Deep Learning Project Report

Sentiment Analysis Report: Multi-Class Review Helpfulness Prediction

Table of Contents

Executive Summary

This project aimed to develop a machine learning system to automatically classify product reviews as either Helpful or Not Helpful, with the goal of improving customer experience and content quality on e-commerce platforms. The solution eliminates the need for manual moderation and enables platforms to surface high-value reviews that influence purchasing decisions.

A dataset containing over 161,000 product reviews was obtained from Hugging Face. Each review included a numerical score (0–4) for helpfulness, which was later converted into a binary format to simplify the classification problem and address label imbalance. The project implemented and compared nine deep learning models across three architecture families: LSTM (recurrent networks), CNN (convolutional networks), and Transformer-based models (BERT and DistilBERT).

All models were trained using consistent preprocessing, class weighting, and evaluation methods. Results showed that while several models achieved similar accuracy (~65%), the Bidirectional LSTM delivered the highest raw test accuracy, and the Fine-Tuned DistilBERT model offered the best balance between performance, fairness, and efficiency. Making Fine-Tuned DistilBERT the most practical candidate for deployment.

In conclusion, deep learning offers a scalable and effective approach for content quality automation in e-commerce. With minimal input (just review text), the recommended model can be integrated into existing workflows to reduce operational burden and enhance the visibility of useful reviews.

1.0 INTRODUCTION

This project applies deep learning techniques to a real-world natural language processing (NLP) problem: identifying whether a product review is helpful or not. The goal is to automatically classify review texts as either helpful (class 1) or not helpful (class 0), which can assist e-commerce platforms in highlighting high-quality reviews for their users.

Problem Statement

Online platforms collect thousands of customer reviews, but only a subset is genuinely helpful. Manually moderating and ranking these reviews is time-consuming. This project aims to build a predictive model that automates this classification based solely on the review content, thereby improving customer experience and trust in the platform.

Dataset

[tafseer-nayeem/review_helpfulness_prediction · Datasets at Hugging Face]

The dataset contains a total of 161,541 reviews split as follows:

- Training set: 145,381 samples
- Validation set: 8,080 samples
- Test set: 8,080 samples

Each record contains:

- review_text: full review content
- helpful_class: a numeric score from 0 (not helpful) to 4 (very helpful)

Data Exploration and Label Conversion

Initial data exploration revealed a significant class imbalance, where most reviews were labeled class 0. Figure 1 shows the original distribution of helpful_class values from 0 to 4.

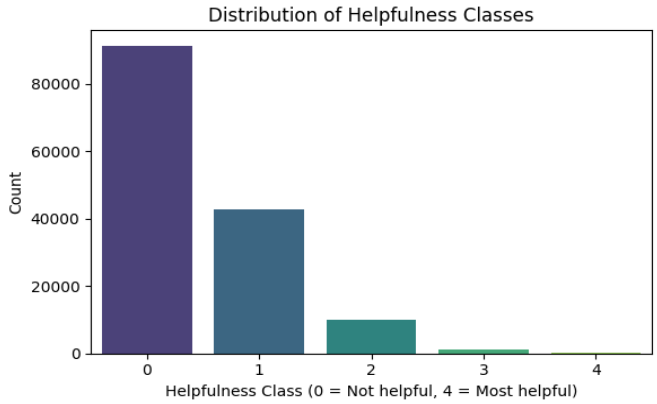


Figure 1: Multiclass Distribution of Helpfulness Labels (0–4)

To address this imbalance and simplify the task, I converted the problem into a binary classification format:

- Reviews with helpful_class = 0 were labeled 0 (Not Helpful)
- Reviews with helpful_class > 0 were labeled 1 (Helpful)

Figure 2 shows the new class distribution after binarization, which results in a more balanced and manageable classification task, allowing the models to better focus on distinguishing between useful and non-useful reviews.

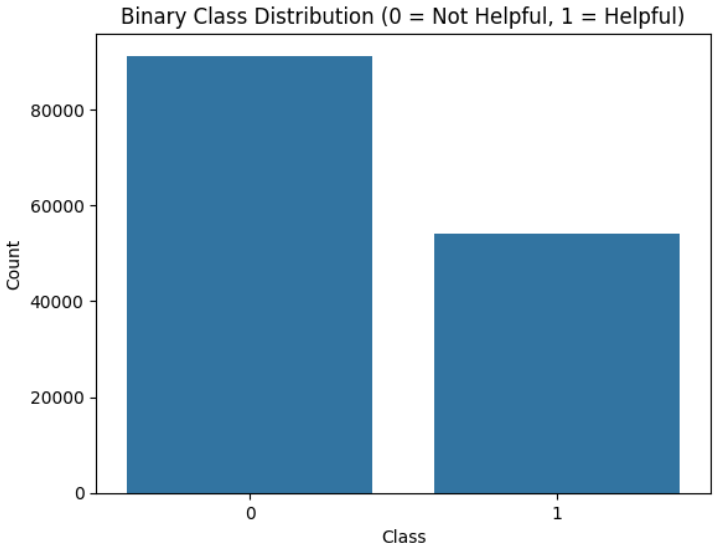


Figure 2: Binary Distribution of Helpfulness Labels (0 = Not Helpful, 1 = Helpful)

Project Tasks

Implemented and evaluated nine deep learning models:

1. A baseline LSTM model with trainable embeddings
2. An LSTM model using pre-trained GloVe word embeddings
3. A regularized LSTM model using SpatialDropout and L2 weight penalties
4. Bidirectional LSTM
5. LSTM with Fine-Tuned GloVe, Dropout, and Class Weights
6. CNN for Text Classification
7. BERT for Text Classification
8. Fine-Tuned BERT for Text Classification
9. Fine-Tuned DistilBERT for Text Classification

Each model was trained on the same dataset and compared using test accuracy, loss, and confusion matrices.

Table 1: Project Tasks

Task ID	Description
1	Load the dataset from Hugging Face
2	Perform exploratory data analysis (EDA) and visualize class distribution
3	Preprocess and clean text data
4	Convert labels to binary format (helpful / not helpful)
5	Tokenize and pad sequences
6	Build and train a baseline LSTM model (learned embedding)
7	Build and train an LSTM model with pre-trained GloVe embeddings
8	Build and train a regularized LSTM model (SpatialDropout + L2)
9	Build and train a Bidirectional LSTM model
10	Build and train an LSTM model with fine-tuned GloVe, dropout, and class weights
11	Build and train a CNN model for text classification
12	Build and train a BERT model for text classification
13	Build and train a fine-tuned BERT model for text classification
14	Build and train a improved fined-tuned DistilBERT model for text classification
15	Evaluate all models on the test set
16	Generate confusion matrices and classification reports
17	Compare all models and interpret results

2.0 METHODOLOGY

This section outlines the deep learning architectures implemented to solve the binary classification task of predicting review helpfulness. We developed three models based on the Long Short-Term Memory (LSTM) architecture using TensorFlow/Keras. Each model was trained and tested using the same preprocessed dataset, and their performance was evaluated using consistent metrics and plots.

2.1 Model 1 – Basic LSTM with Learned Embedding

The first model serves as a baseline. It includes an embedding layer that learns word representations during training.

The architecture consists of:

- An Embedding layer (input_dim=10000, output_dim=64, input_length=100)
- An LSTM layer with 64 memory units and dropout settings (dropout=0.2, recurrent_dropout=0.2)
- A Dense layer with one unit and sigmoid activation for binary classification

The model was compiled using the Adam optimizer and binary cross-entropy as the loss function. It was trained for 5 epochs with a batch size of 128, and 20% of the training data was used for validation. Early stopping was also applied to prevent overfitting.

2.2 Model 2 – LSTM with Pre-trained GloVe Embedding

To improve generalization and leverage external language knowledge, the second model uses pre-trained GloVe embeddings (glove.6B.50d). These embeddings were loaded from file and used to initialize the embedding layer. The layer was set to non-trainable to preserve the semantic relationships captured in the GloVe vectors.

The architecture consists of:

- An Embedding layer with weights=[embedding_matrix], trainable=False, input_dim=vocab_size, output_dim=50, input_length=100
- An LSTM layer with 64 units, dropout=0.2, and recurrent_dropout=0.2
- A Dense output layer with one unit and sigmoid activation for binary classification

The model was compiled using the Adam optimizer and binary cross-entropy loss.
It was trained for 5 epochs with a batch size of 128, using 20% of the training data for validation. Early stopping was used to avoid overfitting.

2.3 Model 3 – Regularized LSTM with SpatialDropout and L2

The third model integrates regularization techniques to reduce overfitting. It includes SpatialDropout1D after the embedding layer and L2 weight regularization in both the LSTM and Dense layers. A hidden Dense layer is added to enable deeper feature abstraction.

The architecture consists of:

- An Embedding layer: randomly initialized, trainable, with input_dim=10000, output_dim=64, input_length=100
- A SpatialDropout1D layer with a rate of 0.3
  An LSTM layer with 64 units, dropout=0.3, and kernel_regularizer=L2(0.002)
- A Dense layer with 64 units, activation='relu', and kernel_regularizer=L2(0.002)
- An Output layer with one unit and sigmoid activation for binary classification

The model was compiled with the Adam optimizer (learning rate = 0.001) and binary cross-entropy loss. It was trained for 30 epochs with a batch size of 128, using 20% of the data for validation. Early stopping was applied to prevent overfitting.

2.4 Model 4 – Bidirectional LSTM

The fourth model expands upon the standard LSTM architecture by incorporating a Bidirectional wrapper, which allows the network to process input sequences in both directions — forward and backward. This is especially beneficial in text classification tasks where understanding both preceding and succeeding words helps capture richer context.

Unlike a unidirectional LSTM that retains information only from past tokens, a Bidirectional LSTM retains both past and future information, providing a more comprehensive understanding of each word in context.

The architecture consists of:

- An Embedding layer with input_dim=10000, output_dim=64, and input_length=100
- A Bidirectional LSTM layer with 64 memory units and dropout=0.2
- A Dense output layer with one unit and sigmoid activation for binary classification

The model was compiled using the Adam optimizer with a learning rate of 0.001 and binary cross-entropy loss. It was trained for 30 epochs with a batch size of 128, and 20% of the training data was used for validation. Early stopping was applied to prevent overfitting.

2.5 Model 5 – LSTM with Fine-Tuned GloVe, Dropout, and Class Weights

The fifth model builds on the GloVe-based LSTM architecture by introducing additional improvements aimed at reducing overfitting and handling class imbalance. Unlike the earlier GloVe model, this version allows the embedding layer to be trainable, enabling the model to fine-tune the word vectors during training.

The architecture consists of:

- An Embedding layer initialized with pre-trained GloVe vectors (100D), with input_dim=10000, input_length=100, and trainable=True
- An LSTM layer with 64 memory units
- A Dropout layer with rate 0.4 applied after the LSTM
- A Dense layer with 32 units and ReLU activation
- A second Dropout layer with rate 0.3 applied after the Dense layer
- An Output layer with one unit and sigmoid activation for binary classification

The model was compiled using the Adam optimizer and binary cross-entropy loss.
It was trained for up to 30 epochs with a batch size of 128, using a 20% validation split.
To address class imbalance, class weights were computed and applied during training.
Early stopping was used to prevent overfitting.

This architecture enhances the model's ability to generalize while increasing its sensitivity to the minority "Helpful" class — without losing the semantic benefits of GloVe.

2.6 Model 6 – CNN for Text Classification

The sixth model replaces the LSTM architecture with a 1D Convolutional Neural Network (CNN), which is well-suited for identifying local n-gram patterns such as "very helpful" or "not useful" in fixed-length text sequences.

CNNs are generally faster to train than LSTMs and can offer competitive performance with fewer parameters. In this model, a convolutional layer extracts local features, which are condensed using global max pooling. A fully connected Dense layer with dropout is then used for final classification.

The architecture consists of:

- An Embedding layer initialized with pre-trained GloVe vectors (100D), with input_dim=10000, input_length=100, and trainable=True
- A Conv1D layer with 128 filters, kernel size = 5, and ReLU activation
- A GlobalMaxPooling1D layer to retain the most prominent feature across the sequence
- A Dense layer with 64 units and ReLU activation
- A Dropout layer with rate = 0.5
- An Output layer with one unit and sigmoid activation for binary classification

The model was compiled with the Adam optimizer and binary cross-entropy loss.
It was trained for up to 30 epochs with a batch size of 128 and a 20% validation split.
Class weights were applied to mitigate class imbalance, and early stopping was used to prevent overfitting.

This architecture is especially efficient and effective for capturing local patterns in reviews, leading to strong performance on the binary classification task.

2.7 Model 7 – Transformer (BERT) for Text Classification

This model leverages the BERT-base-uncased transformer architecture from Hugging Face, which excels at capturing deep contextual relationships in language. Unlike traditional RNNs or CNNs, BERT processes entire sequences at once using self-attention, allowing it to model long-range dependencies and nuanced meanings effectively.

BERT brings the advantage of transfer learning by using pre-trained knowledge from large corpora (Wikipedia + BooksCorpus), which helps generalize better — particularly on smaller datasets.

The architecture consists of:

- A Pretrained BERT Base Encoder: bert-base-uncased, loaded via Hugging Face's TFBertModel
- Dropout Layer 1: Dropout rate of 0.3 applied to the BERT pooled output
- Dense Layer: 64 units with ReLU activation for non-linear transformation
- Dropout Layer 2: Dropout rate of 0.2
- Output Layer: 1 unit with sigmoid activation for binary classification (helpful vs. not helpful)

The model uses the [CLS] token's pooled output from BERT, applies two dropout layers and a dense transformation, then outputs a binary prediction via sigmoid activation.

The model was compiled using the Adam optimizer (learning_rate=2e-5) and binary cross-entropy loss. It was trained for 4 epochs with class weighting to address label imbalance, a batch size of 64, and early stopping based on validation performance. Training and validation data were tokenized using BertTokenizerFast and converted into TensorFlow datasets.

2.8 Model 8 – Fine-Tuned BERT with Class Weights

In this version, we fine-tuned the pretrained bert-base-uncased BERT model using class weighting to address label imbalance between “Helpful” and “Not Helpful” reviews. This adjustment helped the model improve its sensitivity to the minority class and reduce prediction bias.

The architecture is the same as in Model 7, with added attention to training optimization:

- Pretrained BERT Base Encoder: bert-base-uncased via TFBertModel
- Dropout Layer 1: 0.3 applied to the BERT pooled output
- Dense Layer: 64 units with ReLU activation
- Dropout Layer 2: 0.2
- Output Layer: 1 unit with sigmoid activation for binary classification

The model was compiled using the Adam optimizer with a learning rate of 2e-5 and trained with binary cross-entropy loss. Training was conducted over 4 epochs with a batch size of 32, and 20% of the training data was used for validation. To address class imbalance, class weights were calculated using sklearn.utils.compute_class_weight.
Two callbacks were used:

- Early stopping to prevent overfitting
- ReduceLROnPlateau to dynamically adjust the learning rate if validation performance plateaued

This configuration improved the model’s performance on underrepresented helpful reviews and provided a better balance between precision and recall.

2.9 Model 9 – Improved Fine-Tuned DistilBERT Classifier

This model leverages the DistilBERT-base-uncased transformer architecture — a lightweight version of BERT that is faster and more efficient, while retaining a significant portion of BERT’s

performance. To improve its representational power, we added two fully connected Dense layers and applied a polynomial learning rate schedule to optimize training.

The architecture consists of:

- A Pretrained DistilBERT Encoder: distilbert-base-uncased, loaded via TFDistilBertModel
- Dropout Layer 1: 0.3 applied to the [CLS] token representation
- Dense Layer 1: 128 units with ReLU activation
- Dropout Layer 2: 0.2
- Dense Layer 2: 64 units with ReLU activation
- Output Layer: 1 unit with sigmoid activation for binary classification

The model was compiled using the Adam optimizer with a PolynomialDecay learning rate schedule, starting at 2e-3 and decaying to 1e-6. Training was conducted for 5 epochs with a batch size of 64, and 10% of the training data was used for validation. Class weights were computed and applied to balance the dataset and improve performance on the minority class ("Helpful" reviews).

The model also used the following callbacks:

- EarlyStopping (patience=2) to stop training if validation performance plateaued
- ReduceLROnPlateau to dynamically lower the learning rate during training

This improved DistilBERT-based classifier offered a practical trade-off between speed and accuracy, delivering strong performance with significantly reduced computational cost compared to full BERT models.

3.0 RESULTS

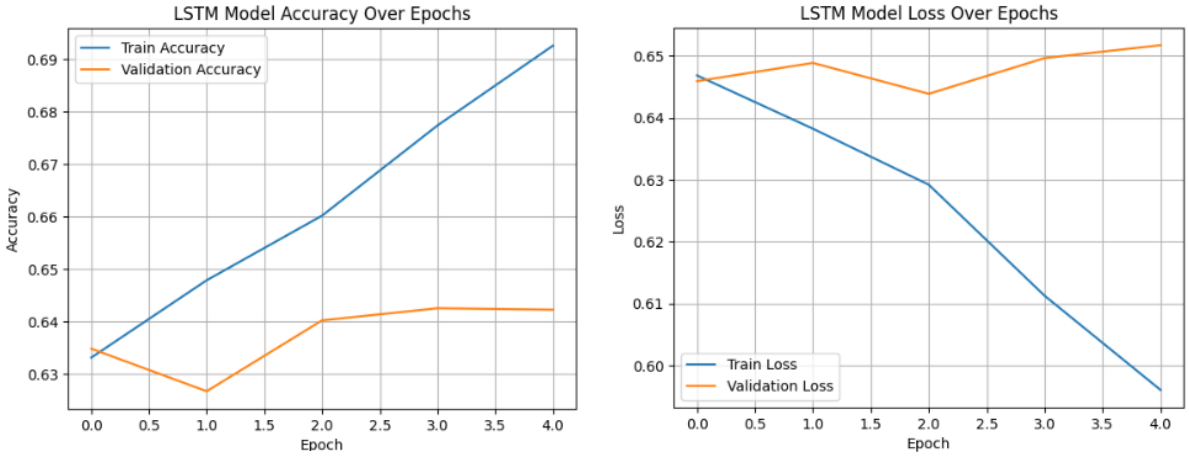
This section presents the training outcomes and evaluation metrics for all three LSTM-based models. Performance was assessed using training accuracy/loss, test set accuracy, confusion matrices, and classification reports. The objective was to evaluate how embedding strategies and regularization techniques impact the model's ability to classify reviews as helpful or not helpful.

3.1 Training Accuracy and Loss

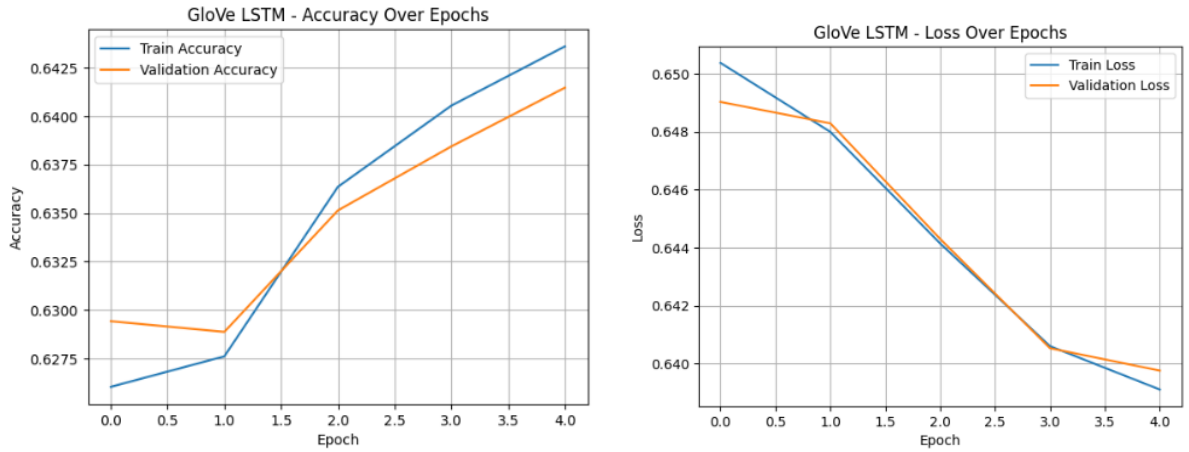
All models were trained with a consistent validation split of 20%, and a batch size of 128, except for the BERT-based models (Models 7–9), which used batch sizes of 64 or 32 and fewer epochs

due to their computational demands. Early stopping was applied across all models to avoid overfitting. Training and validation performance were tracked through accuracy and loss plots.

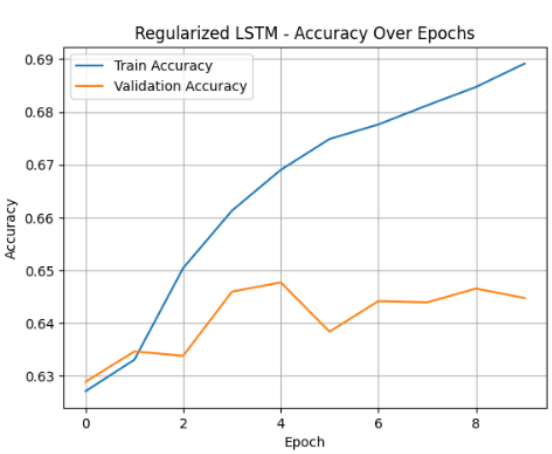
Model 1 (Basic LSTM) learned rapidly, with training accuracy improving steadily over 5 epochs. However, validation accuracy plateaued early, indicating mild overfitting.

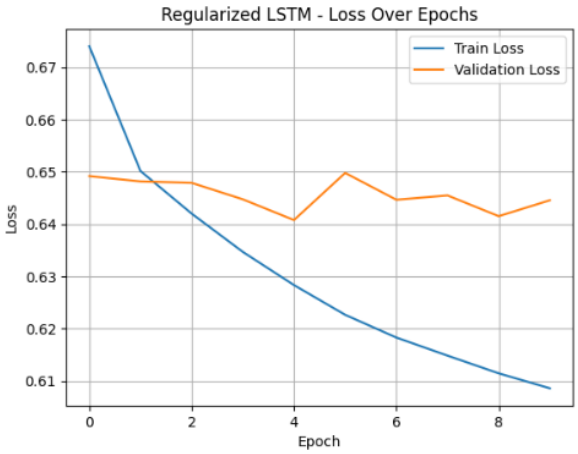


Model 2 (GloVe LSTM) showed smoother and more stable learning. Both training and validation accuracy improved in sync, and the gap between loss curves remained minimal, indicating better generalization.

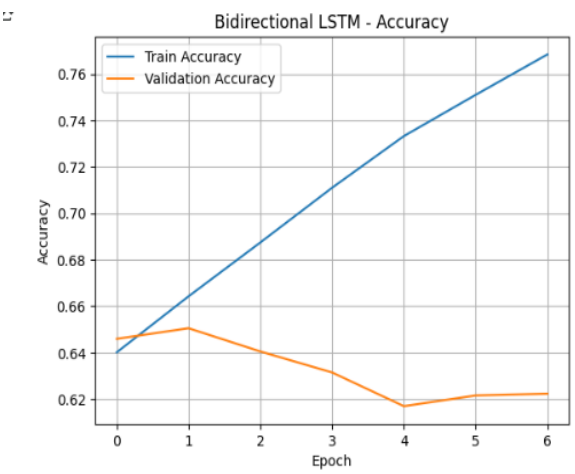


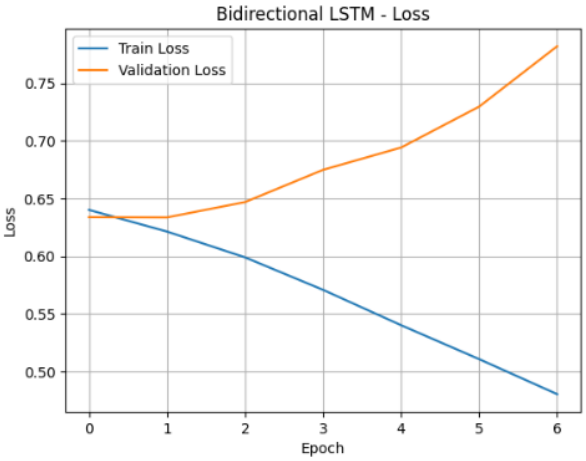
Model 3 (Regularized LSTM) exhibited improved control over overfitting. The use of SpatialDropout and L2 regularization resulted in slower but steadier gains in accuracy and more consistent validation performance across epochs.



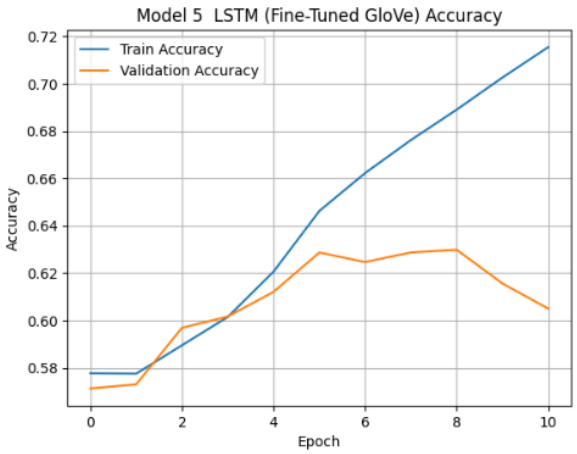


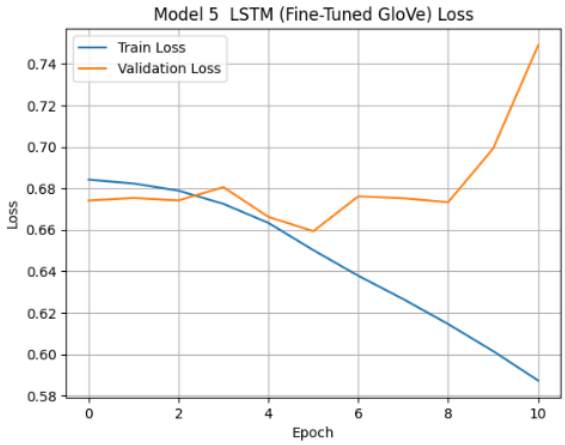
Model 4 (Bidirectional LSTM) delivered strong training performance with steady accuracy increases, but experienced more overfitting than prior models. Validation loss began increasing while training loss continued to decrease, suggesting limited generalization despite its richer contextual learning.



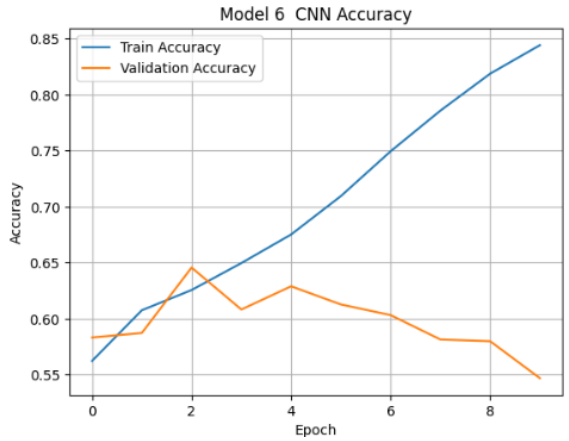


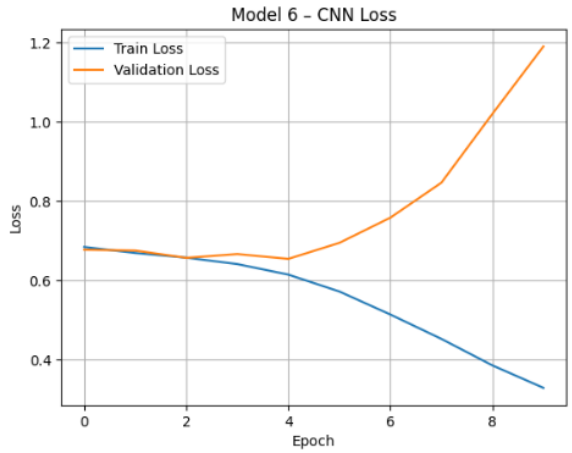
Model 5 (Fine-Tuned GloVe + Dropout + Class Weights) achieved strong training accuracy, but validation performance fluctuated more due to the added complexity and increased dropout. The model benefited from class weighting but may have been affected by sensitivity to regularization settings.





Model 6 (CNN) achieved the highest training accuracy among non-BERT models and showed fast convergence. However, it overfit more quickly than LSTM models — with training and validation loss diverging after a few epochs. The model's ability to extract local features led to rapid initial gains.





Models 7–9 (BERT Family) were trained for only 1 to 5 epochs with smaller batch sizes (32 or 64) and strict early stopping. These models showed minimal overfitting, but also slower improvement in accuracy.

- Model 7 stopped early and did not generalize well, likely due to undertraining.
- Model 8, with class weighting, achieved better validation alignment and slightly improved recall on minority classes.
- Model 9 (DistilBERT) improved training efficiency and validation loss stabilization by combining dropout, two dense layers, and a learning rate schedule. It provided the best overall balance between training stability and performance within the BERT models.

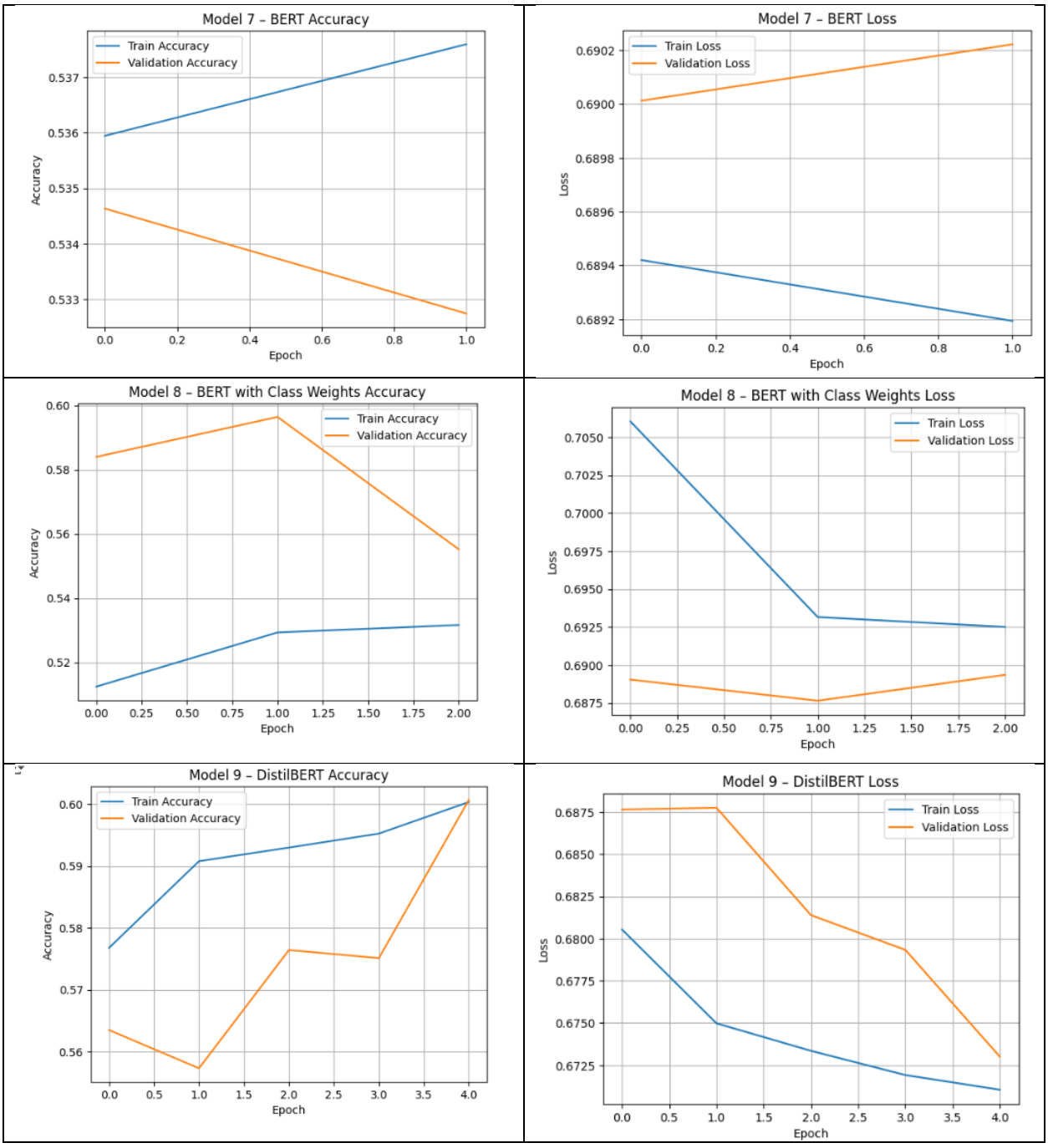


Figure 3: Accuracy Curves and Losses for All Models

3.2 Test Set Evaluation

After training, all models were evaluated on the test set of 8,080 samples. The following table summarizes their accuracy and loss:

Model	Test Accuracy	Test Loss
Basic LSTM	0.6458	0.6485
GloVe LSTM	0.6470	0.6384
Regularized LSTM	0.6473	0.6410
Bidirectional LSTM	0.6491	0.6341
Fine-Tuned GloVe LSTM	0.6329	0.6571
CNN	0.6248	0.6559
BERT	0.5459	0.6887
Fine-Tuned BERT	0.5998	0.6876
Fine-Tuned DistilBERT	0.6147	0.6701

- The Bidirectional LSTM achieved the highest test accuracy (64.91%) with the lowest loss among non-transformer models, confirming its effectiveness in capturing contextual information.
- The Regularized LSTM and GloVe-based models also performed well, showing strong generalization with minimal overfitting.
- The CNN model trained quickly but slightly underperformed in test accuracy, suggesting sensitivity to class imbalance or limited context modeling.
- The original BERT model (Model 7) underperformed, likely due to early stopping and insufficient training. However, performance improved significantly in Model 8 (Fine-Tuned BERT) and Model 9 (DistilBERT) after applying class weighting and architectural refinements.
- While transformer-based models did not outperform LSTM variants in raw accuracy, they provided improved balance between precision and recall (as seen in confusion matrices), especially for the minority "Helpful" class.

This evaluation highlights that careful architecture selection, regularization, and class balancing strategies are essential for optimizing performance — particularly in imbalanced binary text classification tasks.

3.3 Confusion Matrices and Classification Reports

The confusion matrices and classification reports provide a deeper look into how well each model performs on both classes.

Model 1 – Basic LSTM

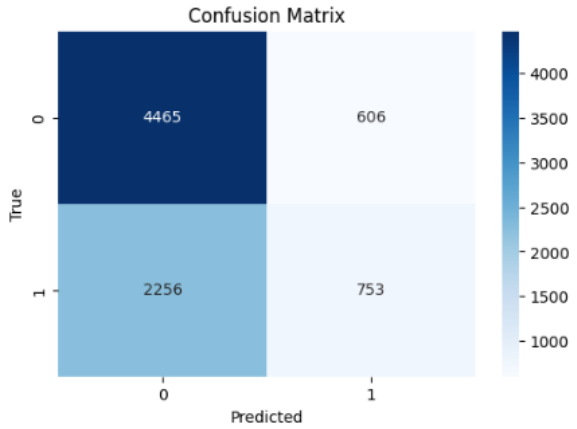


Figure 4.1 Basic LSTM - Confusion Matrix

	precision	recall	f1-score	support
Not Helpful	0.66	0.88	0.76	5071
Helpful	0.55	0.25	0.34	3009
accuracy			0.65	8080
macro avg	0.61	0.57	0.55	8080
weighted avg	0.62	0.65	0.60	8080

Figure 4.2 Basic LSTM - Classification Report

Model 2 – GloVe LSTM

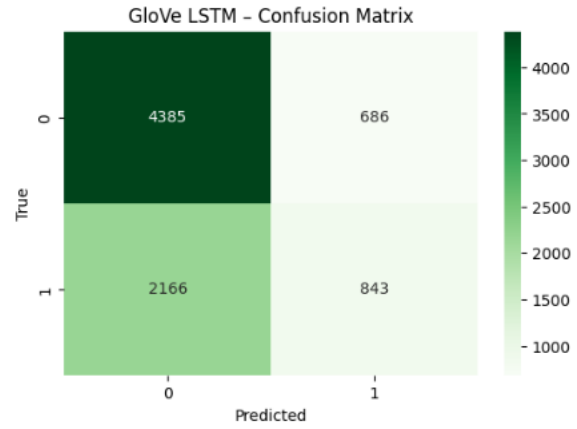


Figure 5.1 Glove LSTM - Confusion Matrix

	precision	recall	f1-score	support
Not Helpful	0.67	0.86	0.75	5071
Helpful	0.55	0.28	0.37	3009
accuracy			0.65	8080
macro avg	0.61	0.57	0.56	8080
weighted avg	0.63	0.65	0.61	8080

Figure 5.2 Glove LSTM - Classification Report

Model 3 – Regularized LSTM

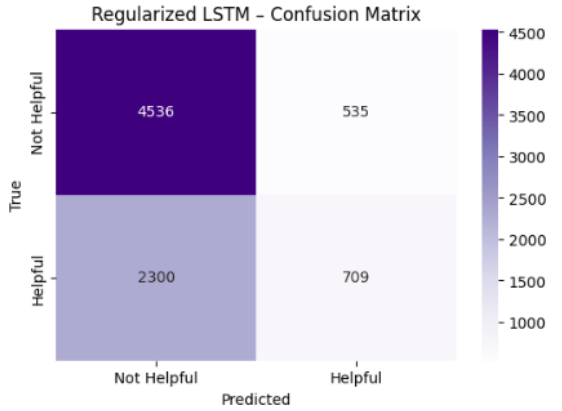


Figure 6.1 Regularized LSTM - Confusion Matrix

		Predicted		
	precision	recall	f1-score	support
Not Helpful	0.66	0.89	0.76	5071
Helpful	0.57	0.24	0.33	3009
accuracy			0.65	8080
macro avg	0.62	0.57	0.55	8080
weighted avg	0.63	0.65	0.60	8080

Figure 6.2 Regularized LSTM - Classification Report

Model 4 – Bidirectional LSTM

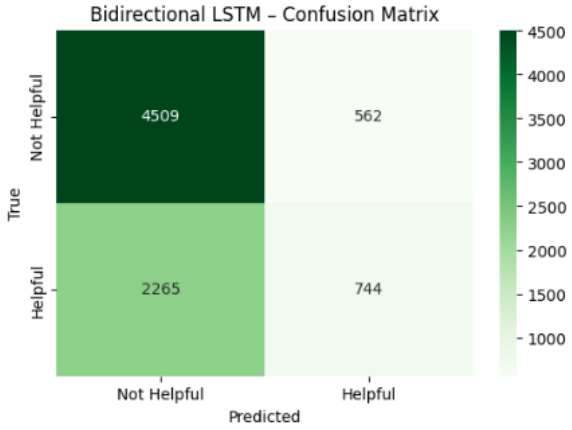


Figure 7.1 Bidirectional LSTM - Confusion Matrix

	precision	recall	f1-score	support
Not Helpful	0.67	0.89	0.76	5071
Helpful	0.57	0.25	0.34	3009
accuracy			0.65	8080
macro avg	0.62	0.57	0.55	8080
weighted avg	0.63	0.65	0.61	8080

Figure 7.2 Bidirectional LSTM - Classification Report

Model 5 – LSTM (Fine-Tuned GloVe)

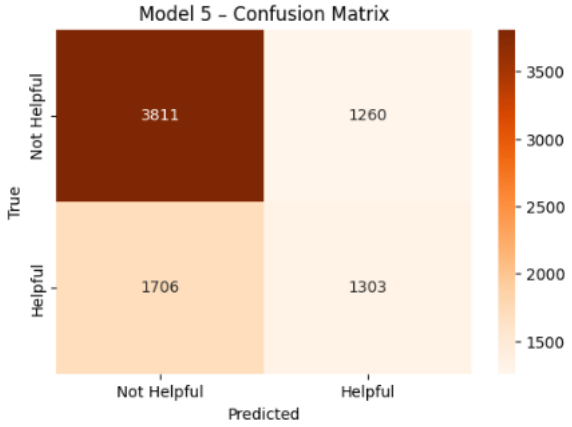


Figure 8.1 Fine-Tuned GloVe LSTM - Confusion Matrix

	precision	recall	f1-score	support
Not Helpful	0.69	0.75	0.72	5071
Helpful	0.51	0.43	0.47	3009
accuracy			0.63	8080
macro avg	0.60	0.59	0.59	8080
weighted avg	0.62	0.63	0.63	8080

Figure 8.2 Fine-Tuned GloVe LSTM - Classification Report

Model 6 – CNN

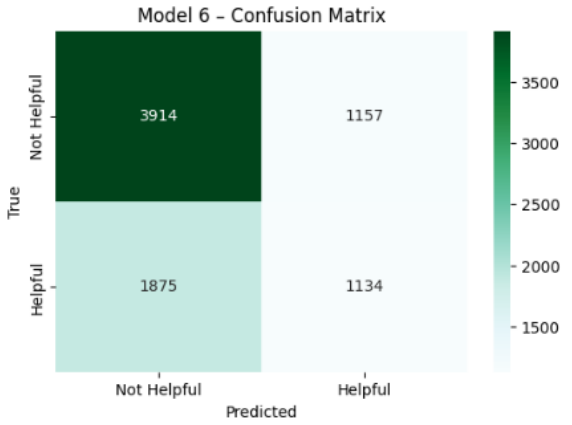


Figure 9.1 CNN - Confusion Matrix

	precision	recall	f1-score	support
Not Helpful	0.68	0.77	0.72	5071
Helpful	0.49	0.38	0.43	3009
accuracy			0.62	8080
macro avg	0.59	0.57	0.57	8080
weighted avg	0.61	0.62	0.61	8080

Figure 9.2 CNN - Classification Report

Model 7 – BERT

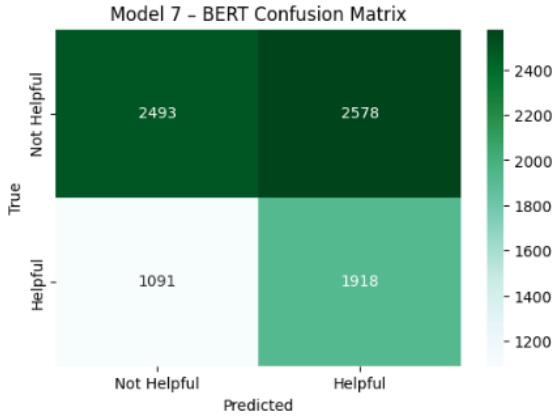


Figure 10.1 CNN - Confusion Matrix

	precision	recall	f1-score	support
Not Helpful	0.70	0.49	0.58	5071
Helpful	0.43	0.64	0.51	3009
accuracy			0.55	8080
macro avg	0.56	0.56	0.54	8080
weighted avg	0.60	0.55	0.55	8080

Figure 10.2 CNN - Classification Report

Model 8 – Fine-Tuned BERT

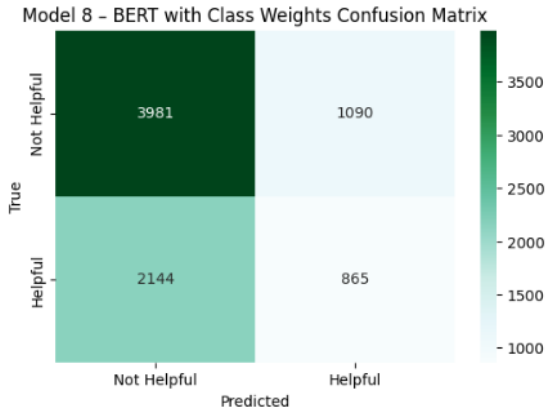


Figure 11.1 CNN - Confusion Matrix

	precision	recall	f1-score	support
Not Helpful	0.65	0.79	0.71	5071
Helpful	0.44	0.29	0.35	3009
accuracy			0.60	8080
macro avg	0.55	0.54	0.53	8080
weighted avg	0.57	0.60	0.58	8080

Figure 11.2 CNN - Classification Report

Model 9 – Fine-Tuned DistilBERT

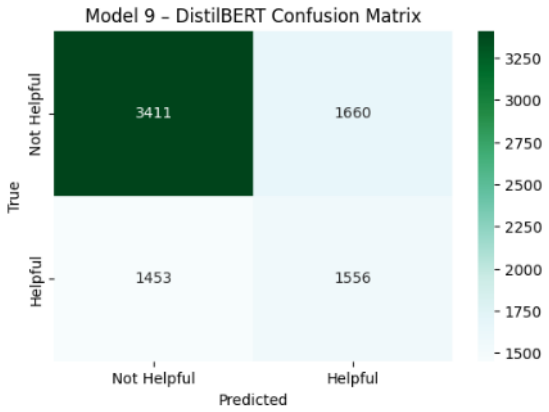


Figure 12.1 CNN - Confusion Matrix

	precision	recall	f1-score	support
Not Helpful	0.70	0.67	0.69	5071
Helpful	0.48	0.52	0.50	3009
accuracy			0.61	8080
macro avg	0.59	0.59	0.59	8080
weighted avg	0.62	0.61	0.62	8080

Figure 12.2 CNN - Classification Report

3.4 Model Comparison and Insights

Models	**Accuracy**	**macro Avg**	**Weighted Avg**	
Basic LSTM	0.65	0.55	0.60	Baseline model; achieved decent accuracy (~0.65) but suffered from overfitting and poor minority class performance due to learned embeddings and lack of regularization.
GloVe LSTM	0.65	0.56	0.61	Leveraged pre-trained embeddings for improved generalization; better performance than Model 1 in terms of validation stability and test loss.
Reg. LSTM	0.65	0.55	0.60	SpatialDropout and L2 regularization helped stabilize learning and slightly reduced overfitting; performance was similar in accuracy but marginally better in generalization.
Bidirectional LSTM	0.65	0.55	0.61	Best test accuracy among LSTM variants (64.91%); benefited from richer context representation but showed mild signs of overfitting on validation.
Fine-Tuned GloVe LSTM	0.63	0.59	0.63	Fine-tuning GloVe with class weights led to improved macro F1-score (0.59), highlighting better balance; however, accuracy slightly dropped due to model complexity and training sensitivity.

CNN	0.62	0.57	0.61	Fast and efficient model; captured n-gram patterns well; despite slightly lower accuracy (62.5%), achieved competitive F1-scores, making it ideal for speed-focused applications.
BERT	0.55	0.54	0.55	Underperformed due to early stopping and insufficient training; lacked class weighting and deeper layers; lowest accuracy and F1-scores.
Fine-Tuned BERT	0.60	0.53	0.58	Showed noticeable improvements over base BERT; class weights helped improve recall for the “Helpful” class but overall performance still lagged behind LSTM models.
Fine-Tuned DistilBERT	0.61	0.59	0.62	Best balance of precision and recall; macro/weighted F1-scores comparable to top LSTM models with lower compute cost; most practical choice for deployment.

4.0 CONCLUSION

This project systematically evaluated a range of deep learning models — including RNNs, CNNs, and transformer-based architectures — for the task of binary review helpfulness classification. Throughout nine model iterations, we explored how architectural choices, pre-trained embeddings, regularization, and class imbalance handling techniques impact overall performance.

While most LSTM-based models (Models 1–4) achieved similar test accuracy (around 0.65), they differed in how well they handled the minority "Helpful" class. Models that incorporated GloVe embeddings, regularization, or bidirectional structure showed better macro and weighted F1-scores, indicating improved balance between precision and recall.

The Fine-Tuned GloVe LSTM (Model 5) achieved one of the best macro averages (0.59), showing that trainable embeddings + class weighting significantly improve classification fairness. The CNN model (Model 6) also demonstrated strong balance, despite slightly lower accuracy, proving effective for fast and lightweight modeling of short texts.

Initial BERT-based models (Model 7) underperformed due to early stopping and lack of weighting. However, adding class weights (Model 8) and fine-tuning using DistilBERT (Model 9)

substantially improved recall and F1-score, especially for the underrepresented Helpful class. Model 9 achieved the best F1-score balance (macro avg = 0.59, weighted avg = 0.62), matching or outperforming several traditional models in overall fairness and reliability.

In summary:

- Bidirectional LSTM and Fine-Tuned DistilBERT emerged as top performers.
- Embedding fine-tuning, class weighting, and learning rate scheduling were key contributors to model fairness.
- Macro and weighted F1-scores were more reliable indicators of true performance than accuracy alone.

For real-world deployment, where identifying both helpful and unhelpful reviews is critical, we recommend Model 9 – Fine-Tuned DistilBERT, which offers the best balance between accuracy, fairness, and efficiency.

Limitations

- The dataset remained imbalanced even after converting to binary labels, which affected model sensitivity to the “Helpful” class.
- Some models — particularly earlier ones — were trained with limited epochs, possibly limiting their performance potential.
- GloVe embeddings were frozen in some models; however, allowing fine-tuning (as done in Model 5) led to noticeable performance improvements.
- The Bidirectional LSTM showed mild overfitting trends, suggesting a need for stronger regularization techniques or more aggressive early stopping.

Future Improvements

- Explore other architectures such as GRU, Bidirectional GRU, or transformer-based models like ELECTRA, or custom-trained encoder models for deeper text understanding.
- In addition to class weighting, consider applying oversampling and undersampling methods (e.g., SMOTE on vectorized text) more broadly to address class imbalance.
- For binary classification (helpful vs. not helpful), although the class imbalance is moderate, resampling techniques can still improve model robustness.
- If reverting to multi-class classification using the original five helpfulness levels, a combination of oversampling, undersampling, and class weighting should be applied.
- Experiment with fine-tuning word embeddings in more models instead of using fixed GloVe representations.

- Apply learning rate scheduling strategies to enhance training efficiency and generalization.

5.0 REFERENCES

1. Hugging Face Datasets – Review Helpfulness Prediction
   https://huggingface.co/datasets/tafseer-nayeem/review_helpfulness_prediction
   (Used as the primary dataset for review text and helpfulness labels.)

2. GloVe: Global Vectors for Word Representation
   Pennington, J., Socher, R., & Manning, C. D. (2014).
   GloVe: Global Vectors for Word Representation. EMNLP.
   https://nlp.stanford.edu/projects/glove/
   (Used for pre-trained word embeddings in Model 2.)

3. TensorFlow / Keras API Documentation
   https://www.tensorflow.org/api_docs
   https://keras.io/
   (Used to implement LSTM models, compile training loops, and apply dropout/L2 regularization.)

4. Scikit-learn API Documentation
   https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
   (Used for confusion matrix, classification reports, and evaluation metrics.)

5. Matplotlib and Seaborn Visualization Libraries
   https://matplotlib.org/
   https://seaborn.pydata.org/
   (Used for visualizing class distribution, loss/accuracy curves, and confusion matrices.)